2019-20 Compiler Design–Summative

Max Gadouleau

Epiphany 2020

Deadline: 2020-03-13. Submission: DUO.

The main aim of this summative is to design and implement a parser for First-Order Logic (FO).

1 Syntax of First-Order Logic

No previous knowledge of FO is required for this assignment. The syntax for FO that we are using in this assignment is given as follows.

Firstly, the symbols are as follows:

- $\mathcal{V}$ is a set of **variables** (e.g. $x_1, x_2, \ldots, x_n$);
- $\mathcal{C}$ is a set of **constants** (e.g. $C, D, \ldots$);
- $\mathcal{P}$ is a set of **predicates** of different arities (e.g. $P(x,y)$ is a predicate of arity 2, $Q(z)$ is a predicate of arity 1) – the arity can be any positive integer;
- $\mathcal{E} = \{=\}$ is the **equality** symbol;
- $\mathcal{L} = \{\wedge, \vee, \implies, \iff, \neg\}$ is the set of logical **connectives**;
- $\mathcal{Q} = \{\exists, \forall\}$ is the set of **quantifiers**.

The last three sets are actually names for the respective items. Obviously, the brackets and the comma are also symbols in this syntax.

Secondly, once the variables, constants, and predicates are defined, the language of all valid formulae is recursively defined as follows.

1. For any predicate $P \in \mathcal{P}$ of arity d and any collection of d variables $y_1, \ldots, y_d \in \mathcal{V}$ (not necessarily distinct), the atom $P(y_1, \ldots, y_d)$ is a valid formula.
2. For any constants $C, D \in \mathcal{C}$ and any variables $x, y \in \mathcal{V}$, the atoms $(C = D)$, $(C = x)$, $(x = C)$ and $(x = y)$ are also valid formulae.
3. If ϕ and ψ are valid formulae, then the following are all valid formulae:
$$(\phi \wedge \psi); \qquad (\phi \vee \psi); \qquad (\phi \implies \psi); \qquad (\phi \iff \psi); \qquad \neg\phi.$$
4. If ϕ is a valid formula, then for any variable $x \in \mathcal{V}$, $\exists x \phi$ and $\forall x \phi$ are also valid formulae.

For instance, for $\mathcal{V} = \{w, x, y, z\}$, $\mathcal{C} = \{C, D\}$ and $\mathcal{P} = \{P, Q\}$ with respective arities 2 and 1, the following is a valid formula:

$$\forall x(\exists y(P(x,y) \implies \neg Q(x)) \vee \exists z(((C = z) \wedge Q(z)) \wedge P(x,z)))$$

2 Objectives

2.1 Input

The input to your program is a text file that includes two things:

1. it specifies the sets $\mathscr{V}$, $\mathscr{C}$, $\mathscr{P}$, $\mathscr{E}$, $\mathscr{L}$, and $\mathscr{Q}$;
2. it describes a (possibly non-valid) FO formula.

For instance, the input file regarding our example (see attached) would be

```
variables: w x y z
constants: C D
predicates: P[2] Q[1]
equality: =
connectives: \land \lor \implies \iff \neg
quantifiers: \exists \forall
formula: \forall x ( \exists y ( P(x,y) \implies \neg Q(x) )
\lor \exists z ( ( (C = z) \land Q(z) ) \land P(x,z) ) )
```

Note that this format allows for blank spaces, tabs and line breaks between symbols in the formula.

2.2 Outputs

Upon reading the input file, your program should return the following outputs.

- The corresponding grammar for the language of valid FO formulae. **(20 marks)**
  It should be formal and give the sets of symbols (terminal and non-terminal) and the set of production rules. It should be concise when listing the rules, e.g. displaying $S \to Sa|Sb|A|\varepsilon$ instead of listing $S \to Sa$, $S \to Sb$, etc.
- The parse tree of a valid formula. **(50 marks)**
  If the formula is valid, then the program should return the parse tree corresponding to that formula. The output format should be an image or a pdf (i.e. something that visualises the tree).
- A log file. **(30 marks)**
  If the input file is of the correct format and if the formula is valid, then the log file should indicate so. Otherwise, the log file should give an error message. Marks will be awarded based on the comprehensiveness, accuracy and clarity of the error messages.

3 Deliverables

Your DUO submission will include the following.

1. The source code of your program.
   It must be written in Python 3.7.4 (the version used on MDS computers) and must run on an MDS computer with Windows 10.
2. Documentation for your program.
   This needs to be short (no more than five pages) but clear and precise. It should give user guidelines on different aspects of your program, such as: how to run your program, the output files, etc. Based on this documentation, I should be able to run your program with my favourite FO formulae and make sense of the outputs I get.
3. (Optional) Examples.
   Optionally, you may include a few (no more than five) worked out examples, each with an input file and its corresponding outputs. You may refer to these examples in your documentation.